الجمهورية الجزائرية الديمقراطية الشعبية
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التكوين والتعليم المهنيين
Ministère de la Formation et de l'Enseignement Professionnels

Centre National de l'Enseignement Professionnel à Distance (CNEPD)

المركز الوطني للتعليم المهني عن بعد (م.و.ت.م.ب)

COURS DE LANGAGE PASCAL

SÉRIE 05

LES STRUCTURES DE DONNÉES

OBJECTIF PÉDAGOGIQUE :

À la fin de cette série, le stagiaire doit être capable de développer un programme pascal.

PLAN DE LA LEÇON :

INTRODUCTION :

Nous avons vu, les types standards du Pascal : INTEGER, REAL, BOOLEAN et CHAR. Dans cette leçon, nous introduisons des types dont nous pouvons nous même définir les propriétés. Ces types augmentent considérablement la puissance de Pascal, ils nous permettent d'écrire des programmes plus clairs et plus efficaces que si nous nous limitons aux types simples.

Ces nouveaux types sont déclarés dans la rubrique définition de types qui se place entre la déclaration de constantes et la déclaration de variables.

I- LES ENSEMBLES :

Un ensemble est une collection d'objets de même type. Le concept d'ensemble est très lié aux mathématiques. C'est un nombre fini de valeurs qui peuvent être ordonnées.

On parle d'un ensemble d'entiers, d'un ensemble de jours de semaine, d'un ensemble de lettres alphabétiques, d'un ensemble de valeurs logiques, ...Etc.

1-Déclaration :

Un ensemble doit avoir pour type de base un type scalaire (non réel), un type intervalle ou un type énuméré. Un type ensemble est déclaré par les mots SET OF:

Syntaxe : < Idf_ type > **= SET OF** < Type de base > **;**

Où < Type de base > est le type des éléments de l'ensemble. Il peut être simple (Integer, Boolean, Char), Intervalle ou type énuméré.

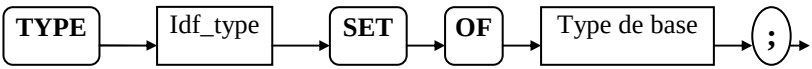


Diagramme de syntaxe

Exemple :

Lettres = SET OF CHAR ; TYPE 1

Chiffres = SET OF 1.. 9 ; TYPE 2

Integer = (POMMES, FRAISES, BANANES, TYPE 3
NOISETTES, CREME, CHOCOLAT, SUCRE GLACE) ;

Dessert = SET OF Integer ;

- Le type Lettres est l'ensemble des caractères
- Le type Chiffres est l'ensemble des chiffres de 1 à 9
- Le type Dessert est l'ensemble dont les éléments sont les valeurs définies dans le type énuméré Integer.

Le type CHAR, le type intervalle 1 .. 9 et le type Integer sont les types de base respectivement des types Lettres, Chiffres et Dessert.

Remarque : Les deux déclarations suivantes sont équivalentes :

```
Chiffres = SET OF 1.. 9;   TYPE        TYPE   Chiffres = 1 .. 9;
C : Chiffres;    VAR                   VAR    C : SET OF Chiffres;
```

Les variables de type ensemble sont déclarées dans la rubrique déclaration de variables de la manière habituelle.

Exemple :

Gouter, Sorbet : Dessert ; VAR
Voyelles : Lettres ;
Num : Chiffres ;

Les valeurs que peut prendre une variable de type ensemble, sont des sous-ensembles de l'ensemble déclaré. Un ensemble est représenté par la liste de ses éléments séparés par des virgules, et entourés par deux crochets.

Exemple :

[‘A’, ‘B’, ‘X’, ‘Y’, ‘Z’] représente l’ensemble des lettres A, B, X, Y et Z;

[1, 2, 3, 4, 5] représente l’ensemble des nombres entiers de 1 à 5;

[X+Y, X*Y, 1-Y] représente l’ensemble des valeurs de ces trois expressions.

Remarque :

1- Si les éléments d’un ensemble sont des valeurs consécutives du type de base, seuls le premier et le dernier ont besoin d’être spécifiés, et donc:

   [Lundi, Mardi, Mercredi, Jeudi] peut s’écrire [Lundi.. Jeudi]

2- Un ensemble peut n’avoir aucun élément du tout. Dans ce cas, c’est l’ensemble vide que l’on note [ ].

3- Si le type de base possède n valeurs, le type ensemble aura 2^n valeurs.

Soit les déclarations suivantes:

```
TYPE  L = SET OF ‘D’.. 'F';
VAR   L1 : L;
```

Il existe $2^3 = 8$ valeurs possibles de L1 et qui sont:

[ ]
[D]
[E]
[F]
[D, E]
[D, F]
[E, F]
[D, E, F]

2- Operations sur les ensembles :

Pascal offre les opérations qui concernent la théorie des ensembles, c'est-à-dire : L'union, l'intersection, la différence et l'appartenance.

2.1- L'union :

L'union de deux ensembles est un ensemble contenant les éléments des deux ensembles. Elle est représentée en Pascal par le signe + (en mathématique∪).

Exemple :

TYPE Véhicule = [Peugeot, Renault, Ford, Citroen, BMW];

VAR A, B: SET OF Véhicule;

Si A = [Peugeot, Renault, Ford]

B = [Citroen, Ford, BMW]

Alors A + B = [Peugeot, Renault, Ford, Citroen, BMW]

2.2- L'intersection :

L'intersection de deux ensembles est un ensemble contenant les éléments communs aux deux ensembles. Elle est représentée par le signe * (en mathématiques ∩).

Exemple :

Si A = [1.. 9]

B = [5.. 20]

Alors A*B = [5.. 9]

2.3- La différence :

La différence entre deux ensembles est un ensemble contenant les éléments du premier ensemble non présents dans le second ensemble. Elle est représentée en Pascal par le signe -.

Exemple :

Si A = [Pommes, Fraises, Bananes]

B = [Fraises, Crème]

Alors A-B = [Pommes, Bananes] et B-A = [Crème]

2.4- L'appartenance :

Elle permet de vérifier si une valeur donnée appartient à l'ensemble défini. Le symbole IN, qui est un mot réservé en Pascal, est utilisé pour tester l'appartenance à un ensemble. L'opérande gauche de IN est une expression, et l'opérande droit est une expression du type ensemble associé à l'opérande gauche.

Syntaxe: < Idf-valeur > **IN** < Idf-ensemble > **;**

Le résultat obtenu est à vrai (TRUE) si la valeur appartient à l'ensemble, sinon il est à faux (FALSE).

Exemple : L'expression Pommes IN [Pommes, Pâtisserie, Sucre] est vraie, mais l'expression Pommes IN [Fraises, Crème] est fausse.

Remarques : Soient S et T deux ensembles, comme pour les opérations mathématiques, les propriétés suivantes restent vraies :

1- S + [ ] donne S
2- S * [ ] donne [ ]
3- S - [ ] donne [ ]
4- [ ] - S donne [ ]

2.5- Egalite, inégalité et inclusion :

Les opérateurs de comparaison peuvent être utilisés pour comparer des ensembles.

- Il est possible d'évaluer les relations d'égalité ou d'inégalité entre deux ensembles. Les opérateurs utilisés sont = et <>.
- Les opérateurs d'inclusion sont <= et >=. Un ensemble X contient un ensemble Y si chaque élément de Y est aussi élément de X.

X <= Y indique X est contenu dans Y.

X >= Y indique X contient Y.

Le résultat de ces comparaisons est bien sûr vrai ou faux.

Exemple :

- [‘A’, ‘B’, ‘C’, ‘D’] <> [‘A’, ‘B’, ‘E’] donne le résultat ture
- [‘A’, ‘B’, ‘C’] = [‘B’, ‘C’, ‘A’] donne le résultat ture
- [1, 2, 3] <= [1, 2, 3, 4, 5] donne le résultat ture
- [1, 2, 3, 4, 5] >= [1, 2, 3] donne le résultat ture

II-L’AFFECTATION :

L'instruction d’affectation peut être utilisée avec des variables de type ensemble et des expressions de même type.

Example:

VAR A: SET OF ‘A’... ‘F’;

A: = [‘B’, ‘E’, ‘D’, ‘C’];

1- Synthèse des opérations :

Opérations		Type du résultat
Union	+	SET OF ...
Intersection	*	SET OF ...
Différence	-	SET OF ...
Appartenance	IN	BOOLEAN
Egalité	=	BOOLEAN
Inégalité	<>	BOOLEAN
Inclusion	<=, >=	BOOLEAN
Affectation	:=	SET OF ...

2-Application sur les ensembles :

```
PROGRAM  Ensemble (OUTPUT);

TYPE  Chiffres = SET OF 1..20;

      Marque = (Peugeot, Renault, Citroën, Ford, BMW, Fiat);
      Véhicule = SET OF Marque;

VAR   A, B, C: Chiffres;

      Voiture, Auto, tout: Véhicule;
      Car: CHAR;
      L1: SET OF 'A'.. 'Z';
      L2: SET OF '0'.. '9';
```

Affectation des ensembles

```
BEGIN

A: = [1.. 9];
B: = [5.. 20 ];
C: = A * B;
```

Intersection des ensembles

```
Voiture: = [Peugeot, Renault];
Auto: = [Citroen, Ford];
Tout: = Voiture + Auto;
```

Union des ensembles

Différence des ensembles

```
C: = B - A;
C: = A -B;

WRITELN ('Entrée un caractère'); READ (Car);
IF (car IN L1) OR (Car IN L2)
WRITELN ('Caractère alphanumérique')     THEN
WRITELN ('Caractère spécial');     ELSE

END.
```

Remarque :

1) Il faut s'assurer de la compatibilité entre les ensembles sur lesquels vont porter les opérations, ils doivent être déclarés de même type.
2) Il ne faut pas définir des ensembles dont la taille ne serait pas acceptée par le compilateur.

3- Les articles :

Un article appelé aussi enregistrement, comme un tableau, est une variable structurée avec plusieurs champs. Mais les champs d'un article peuvent avoir des types différents, et sont accessibles par leur nom et non pas par un indice. Les composants d'un article sont appelés **champs**.

La description d'une planète par un article pourrait être faite à l'aide des informations suivantes, et qui représentent ses champs :

Nom
Visible à l'œil nu (Oui/Non)
Diamètre
Rayon de l'orbite

3.1- Déclaration :

La déclaration d'un enregistrement en Pascal se fait par la déclaration de ses champs, et qui est comprise entre les mots réservés RECORD et END respectivement pour le début et pour la fin.

Syntaxe: **TYPE** <Identificateur> **= RECORD**

<Champ$_1$>**:** <Type$_1$>**;**
<Champ$_2$>**:** <Type$_2$>**;**
.
.
.
<Champ$_n$>**:** <Type$_n$>**;**
END;

Où <Champi> i=1, n est l'indentificateur du i^{ème} champ et qui peut être une liste de champs séparés par des virgules.

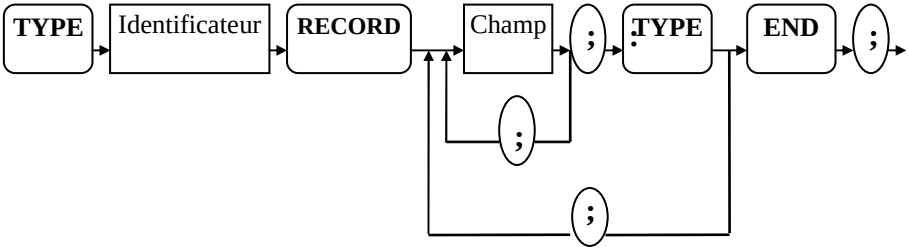


Diagramme de syntaxe

Exemple :

Le type Planète est déclaré comme suit :

```
TYPE   Planète = RECORD
                    Nom: PACKED ARRAY [1..7] OF CHAR;
               Visible: BOOLEAN;
                    Diamètre, Orbite: REAL;
               END;
```

Cet exemple illustre la manière de déclarer un article qui comporte un champ de type chaîne de caractères, un champ booléen et deux champs réels.

La définition d'une variable de type article est semblable à la définition d'une variable d'un autre type. La déclaration d'un type est entourée par les mots clés RECORD et END. Le point-virgule entre REAL et END dans l'exemple précédent peut être omis.

Les variables de ce type sont déclarées de la façon habituelle :

```
VAR   Proche, Lointaine: Planète;
```

Remarque :

1- Le type d'un champ dans un article donné peut être de type quelconque y compris article, sauf de type fichier.
2- Les noms des champs doivent être uniques à l'intérieur d'un article.
3- Nous pouvons utiliser le même nom de champ dans plusieurs articles différents.
4- Les instructions READ et WRITE de Pascal peuvent être utilisées directement pour lire ou écrire les champs d'un article.
5- Un article peut être passé comme paramètre d'une procédure ou d'une fonction.

3.2- Utilisation des articles :

On accède au champ d'un article par **le nom de la variable** article et **le nom du champ**, séparés par **un point**.

Exemple :

Proche. Nom est le nom de la Planète Proche ;
Lointaine. Diamètre est le diamètre de la planète Lointaine.

Ces noms sont appelés sélecteurs, et sont utilisés dans un programme exactement de la même façon que des variables du même type. Nous pouvons écrire les affectations suivantes:

```
Proche. Nom := 'VENUS  ';
Proche. Visible := TRUE;
Lointaine. Orbite := 4496.6;
```

Remarque :

1- La valeur d'un article donné peut être affectée à un autre article par une instruction d'affectation. Par exemple l'instruction Proche := Lointaine est équivalente aux affectations :

```
Proche. Nom := Lointaine. Nom;
Proche. Visible := Lointaine. Visible;
Proche. Diamètre := Lointaine. Diamètre;
Proche. Orbite := Lointaine. Orbite;
```

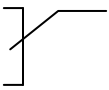
2- Puisque le type d'un champ de l'article peut être lui-même un article, donc Pascal permet l'imbrication des articles : C'est à dire la déclaration d'un article à l'intérieur d'un autre article. Ainsi l'instruction READ (Elève. Datenais. Année) réalise la lecture de l'année de naissance d'un élève donné.

Ici Elève est de type article, dont le champ Datenaiss est lui-même un article, et la déclaration de l'article Datenaiss doit se faire avant celle de l'article Elève.

Exemple :

Dans cet exemple, nous allons voir la déclaration des articles imbriqués ainsi que l'accès à leurs champs.

```
PROGRAM Article (INPUT, OUTPUT);
TYPE  Date = RECORD
        Jour : 1..31;
        Mois: 1..12;
              Année: INTEGER;
             END;

        Adresse = RECORD
              Numéro: INTEGER;
              Ville, chemin: PACKED ARRAY [ 1..20] OF CHAR;
        Codpos: INTEGER;
        Pays: PACKED ARRAY [1..15] OF CHAR;
                 END;

        Enf = RECORD
              Prénom: PACKED ARRAY [1..15] OF CHAR;
              Datenais: Date;
        Sexe: (F, M);
             END;

        Employé = RECORD
        Nom, Prénom: PACKED ARRAY [1..15] OF CHAR;
              Datenais: date;
              Adr perso, A drprof: Adresse;
              Dat entrée: Date;
        Nb enfant: INTEGER;
```



Champs article

```
                Enfants: ARRAY [1..10] OF Enf;
                  END;

VAR             E: Employé;
                I: INTEGER;

BEGIN                   (* Voici quelques exemples d'utilisation *)
WRITELN ('Entrer le nom d'un employé');
READLN (E. Nom);
WRITELN ('Entrer le mois de son recrutement');
READLN (E. Datentrée. Mois);
WRITELN ('Entrer le nombre de ses enfants');
READLN (E. Nb enfant);
FOR I: = 1 TO E. Nb enfant
        DO READLN (E . Enfant [I] . Datenais . Année);

END.
```

3.3- Article avec variantes :

Un enregistrement avec variantes, se compose de deux parties : Une partie fixe et une partie variable. La partie fixe est un ensemble de champ commun à tous les enregistrements du même type et la partie variante est un champ qui varie d'un enregistrement à l'autre.

Exemple :

Le type Personne dont les champs son Nom, Prénom, Etat civil est un article avec variantes, cependant la partie fixe correspond aux champs Nom et Prénom et la partie variante correspond au champ Etat civil qui prend des possibilités différentes suivant la valeur qui lui correspond.

L'état civil d'une personne peut prendre quatre possibilités : CELIBATAIRE, MARIE, VEUF, DIVORCE. Pour chacune d'entre elles, une information spécifique est demandée.

- à CELIBATAIRE ne nécessite aucune information ;
- à MARIE nécessite : la date de mariage, le prénom du conjoint et le nombre d'enfants ;

- à VEUF nécessite : le nombre d'enfants ;
- à DIVORCE nécessite : la date de divorce et le nombre d'enfants.

La déclaration d'un enregistrement avec variantes se fait au moyen de CASE, par la syntaxe suivante :

<Identificateur> **= RECORD**
<Champf$_1$> **:** <Typef$_1$>**;**
<Champf$_2$> **:** <Typef$_2$>**;**
.
.
.
<Champf$_n$> **:** <Typef$_n$>**;**
CASE <Champ$_v$> **:** <Type$_v$> **OF**
V_1**: (** Liste$_1$ des champs **);**
V_2**: (** Liste$_2$ des champs **);**
.
.
.
V_k**: (** liste$_k$ des champs **);**
END;

Où

- <Champf$_1$>, <Champf$_2$>, ..., <Champf$_n$> sont les identificataurs des champs fixes
- <Typef$_1$>, <Typef$_2$>, ..., <Typef$_n$> sont respectivement leurs types;
- <champ$_v$> est l'identificateur du champ variable;
- <Types$_v$> est son type;
- V_1, V_2, ... , V_k sont les valeurs possibles de <champ$_v$>;
- et (Liste$_1$ des champs), (Liste$_2$ des champs) , ... ,
- (Liste$_k$ des champs) sont des listes différentes des champs correspondant respectivement à V_1, v_2, ..., V_k.

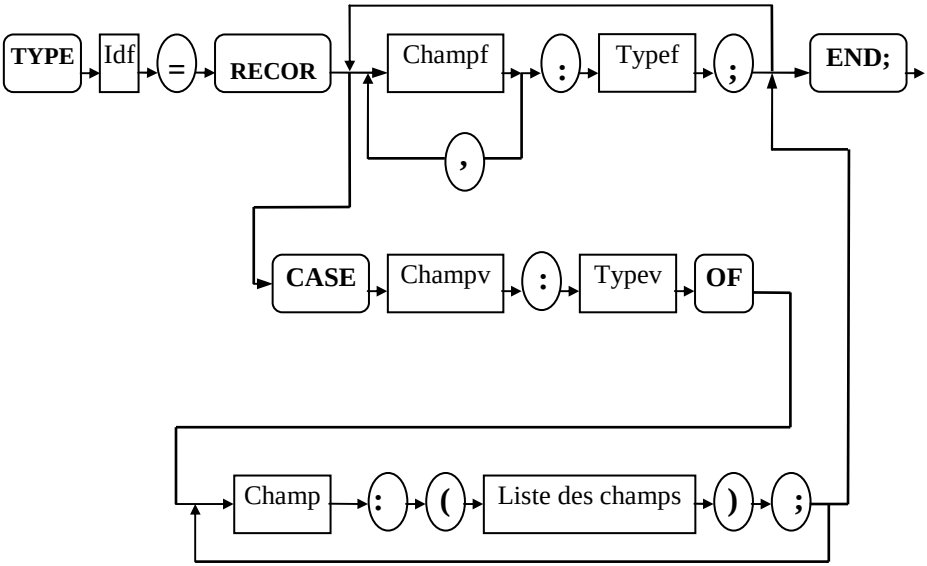


Diagramme de syntaxe

Si nous revenons à l'exemple précédent, la déclaration de l'article personne ne se fait comme suit :

```
TYPE  Etat = (Célibataire, Marié, Veuf, Divorcé) ;
      Date = RECORD
               JJ: 1..31 ;
               MM: 1..12 ;
               AA: INTEGER;
               END ;

      Personne = RECORD
                   Nom, Prénom : PACKED ARRAY [ 1..15] OF
CHAR ;
                   CASE  Etacivil : Etat OF
Célibataire : ( ) ;

Marié : ( Datem : Date;
Prénom : STRING [15];
Nbenf : INTEGER ) ;
Veuf : ( Nbenfan : INTEGER ) ;
```

Divorcé : (Datediv : Date) ;
Nbenfant : INTEGER) ;
END ;

Si P1, P2, P3 sont des variables de type Personne, Elles peuvent avoir pour valeurs :

P1 :	KADOUR NASSIM 01/06/87	P2 :	CHABANI NADIA AHMED 1	P3 :	LOULI REDHA 20/02/97

Ce qui signifie que : KADOUR NASSIM est célibataire ;
CHABANI NADIA est mariée à AHMED le 01/06/87 et elle est mère à un enfant ;
LOULI REDHA est divorcé le 20/02/97 et est père à deux enfants.

Remarque :

1- Un article ne peut avoir qu'une seule partie variante et qui doit obligatoirement être déclarée après la partie fixe.
2- Si pour un cas donné, la liste de champs est vide (cas de Célibataire dans l'exemple précédent), il faut malgré tout écrire : Identificateur de cas : ( ) ;
3- Les identificateurs employés dans les différentes variantes (liste de champs) doivent être uniques à l'intérieur de l'article : Un même identificateur de champs ne peut être employé dans deux variantes d'un article, ni être utilisé simultanément dans la partie fixe et la partie variable.
4- Toutes les valeurs possibles d'un champ variable doivent apparaître comme des constantes.
5- Le compilateur Pascal réserve un emplacement mémoire qui correspond à la variable de plus grande taille.
6- Il est possible d'avoir dans un article plusieurs niveaux de variantes.

Exemple :

Cet exemple illustre l'utilisation d'articles avec variantes :

```
PROGRAM  Articlev ( INPUT, OUTPUT ) ;

TYPE  Date = RECORD
                    JJ : 1..31 ;
                    MM : 1..12 ;
                    AA : INTEGER ;
                     END ;
Emploi = (Secrétaire, Dactylo, Comptable) ;
Langue = (Anglais, arabe, Allemand) ;
Employé = RECORD
Nom, Prénom : STRING [15] ;
CASE  Travail : Emploi  OF
Secrétaire : (Datenais : Date ;
Salaire : REAL ;
CASE  Niveau : Langue OF
Anglais : (Lu, Ecrit, Parlé : BOOLEAN) ;
Arabe: (L, Parl : BOOLEAN) ;
Allemande: (Par: BOOLEAN);
Dactylo : (Daten : date ; salair : REAL ; Niv : INTEGER) ;
Comptable: (Salai : REAL; Ancien : INTEGER) ;
                END;

VAR    Elément: Employé;

BEGIN
Elément. Nom: = 'ABED';
Elément. Prénom := 'MOHAMED' ;
Elément. Travail := Secrétaire ;
Elément. Datenai . JJ := 24 ;
Elément. Datenai . MM := 10 ;
Elément. Lu := TRUE ;
END.
```

CASE Travail : Emploi OF est le premier niveau spécifiant des caractéristiques pour la catégorie de l'emploi secrétaire.

CASE Niveau : Langue OF est le deuxième niveau spécifiant des caractéristiques liées à l'utilisation de la langue.

III- LES POINTEURS :

Les différentes structures de données, telles que celles vues dans les leçons précédentes, sont dites statiques, c'est à dire qu'elles sont déclarées explicitement dans un programme (ou sous-programme) et définies par un identificateur. La taille de telles variables est connue et est définie dans la partie déclarations du programme.

Mais des variables peuvent être générées dynamiquement, c'est à dire en cours d'exécution, au fur et à mesure des besoins, alors que leur nombre n'est pas connu à priori. Ces variables sont appelées variables dynamiques.

Les variables dynamiques ainsi générées n'ont pas de nom, elles sont repérées, donc accessibles à l'aide de pointeur.

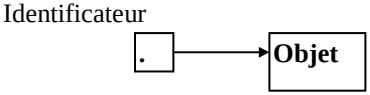
1- Définition et syntaxe du type pointeur :

Le type pointeur est un type simple comme INTEGER, REAL, ou BOOLEAN. « Pointeur » n'est cependant pas un identificateur prédéfini.

La déclaration du type pointeur se fait au moyen du symbole↑.

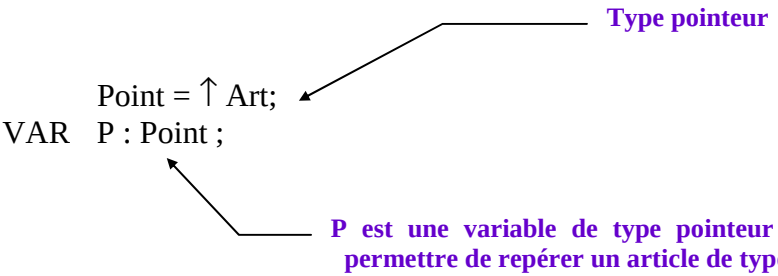
Syntaxe : **TYPE** <Identificateur> = ↑ <Objet> **;**

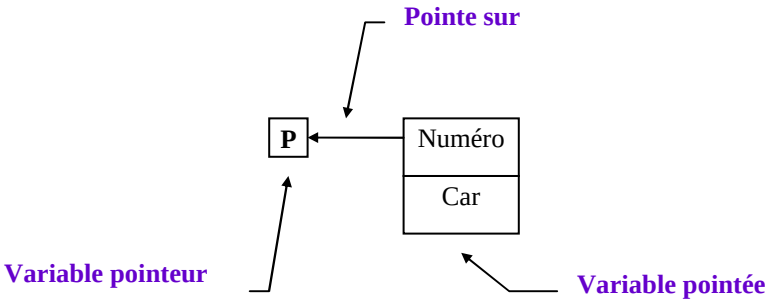
qui indique que identificateur est un type pointeur. Une variable de type Objet peut être associée à un pointeur de type identificateur, ce qui est schématisé par :



Exemple :

```
PROGRAM P1;
TYPE  Art = RECORD
              Numéro: INTEGER;
              Car: CHAR;
            END;
```





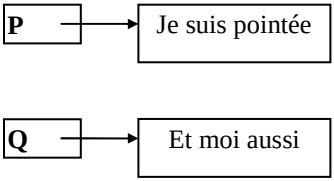
Donc, la variable pointeur P contient l'adresse mémoire de la variable pointée, qui est l'article (Numéro, Car).

À la place de Type Point = ↑ art ;
VAR P : Point ;

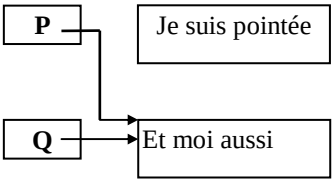
Nous aurions pu écrire VAR P : ↑ Art ;

Remarque :

1- La durée de vie d'une variable pointeur est indépendante de celle du bloc de programme dans lequel elle est déclarée, c'est à dire qu'elle est allouée et désallouée par l'appel de procédures prédéfinies (NEW et DISPOSE).
2- Une variable pointeur qui n'est associée à aucun objet prend la valeur NIL, qui est un mot réservé en Pascal.
3- Il est important de faire la distinction entre les pointeurs et les choses qu'ils pointent. Dans la figure suivante le pointeur P pointe la chaîne de caractères « Je suis pointée », et le pointeur Q pointe la chaîne de caractères « Et moi aussi ».

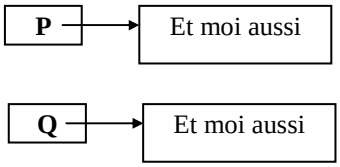


- L'affectation P := Q ; a pour effet d'affecter la valeur du pointeur Q au pointeur P, ce qui est illustré par :



Maintenant la chaîne de caractère « Je suis pointée » ne peut être accédée par le programme

- L'instruction P ↑ := Q ↑ ; a pour effet tout à fait différent. Elle recopie la valeur de l'objet Q ↑ dans l'objet P ↑, ce qui est illustré par :

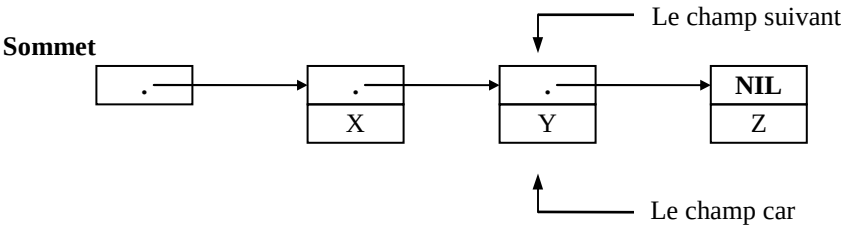


L'affectation de deux variables pointeurs n'est autorisée que si les deux variables sont de même type.

Une structure de donnée dynamique est constituée d'un certain nombre d'éléments reliés par des pointeurs. On peut ajouter de nouveaux éléments à la structure ou en supprimer d'autres pendant l'exécution du programme.

Chaque élément d'une structure de donnés dynamique peut contenir un ou plusieurs champs pointant vers d'autres éléments. Ainsi un élément pointé pourrait être de type simple, pointeur, tableau ou enregistrement.

La liste chaînée est la plus simple des structures de données dynamiques. Le schéma ci-dessous montre une structure de données construite à partir d'un seul pointeur, Sommet et de trois éléments de type article.



Le type d'un élément de cette liste peut être déclaré par :

```
TYPE  Pteur = ↑ Art
      Art = RECORD
              Suivant : Pteur ;
              Car : CHAR ;
            END ;
```

Pointeur qui fait le lien avec l'élément

Contient un caractère

Le dernier élément de la liste a la valeur (NIL, Z).

Remarque : Pour accéder à un champ d'une variable pointeur, il faut utiliser la notation pointée (↑), c'est à dire le nom de la variable pointeur et celui de son champ, séparés par le symbole ↑ suivi du point.

Exemple :

Soit Elément une variable de type pointeur déclaré ci-dessus.

Elément↑**.** Suivant donne la valeur du champ **Suivant ;**

Elément↑**.** Car donne la valeur du champ **Car.**

Elément↑**.** Suivant := NIL ; affecte la valeur NIL au champ Suivant ;

Elément↑**.** Car := 'A'; affecte la valeur 'A' au champ Car.

2- Création d'une variable dynamique : NEW

Pour créer dynamiquement une variable, on utilise la procédure standard prédéfinie NEW, dont la syntaxe est :

Syntaxe : NEW (<Identificateur>**) ;**

Où <Identificateur> est le nom de la variable pointeur à créer.

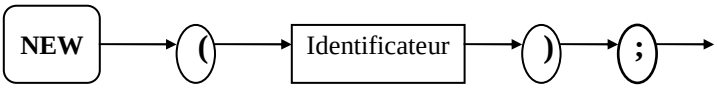


Diagramme de syntaxe

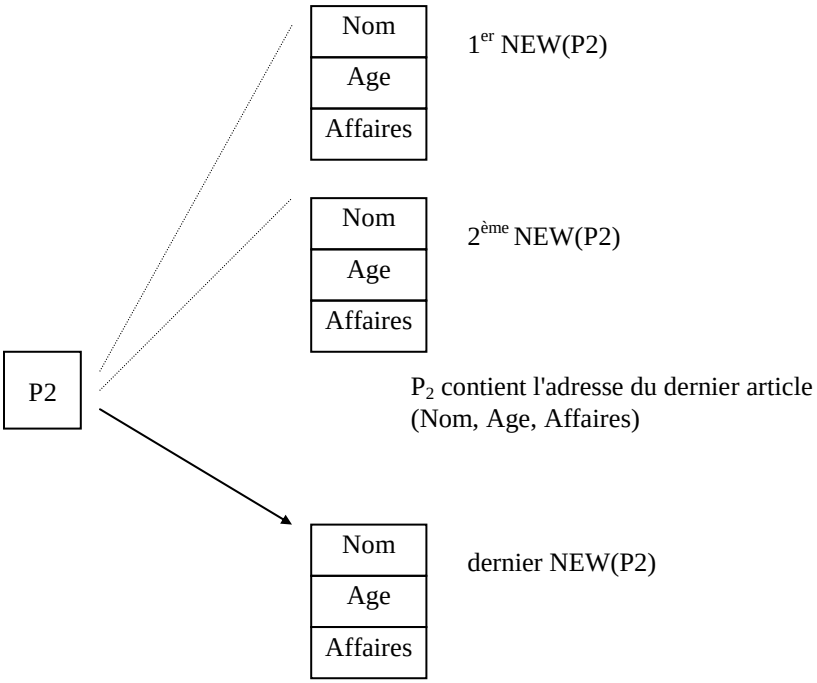
L'exécution de l'instruction NEW (T) permet l'allocation d'un espace mémoire qui correspond à un élément de type pointé par la variable pointeur T.

Exemple:

```
PROGRAM P_New ;

TYPE  Tab = ARRAY [ 1..10 ] OF INTEGER ;
      Personne = RECORD
                    Nom: STRING [20];
                    Age: INTEGER;
                    Affaires: REAL;
                 END;

VAR   P1 : ↑ Tab ;
      P2 : ↑ Personne ;

BEGIN

NEW(P1) ;     (* Alloue la place mémoire à un composant de type
Tab-10 éléments de type entier- *)


NEW(P2) ;        (* Alloue la place mémoire à un composant de type
Personne -une chaîne de 20 caractères + un entier    + un réel- *)
END.
```

Remarque :

Si nous avions répété successivement l'instruction NEW (P2), il y aurait création de plusieurs variables pointées (Personne), mais attention, seule la dernière valeur adresse est conservée dans P2. (Les précédentes sont écrasées par les nouvelles au fur et à mesure).



3- Désaffectation d'une variable dynamique : DISPOSE

Pour libérer la place mémoire allouée à la variable pointée, on utilise la procédure standard prédéfinie DISPOSE, dont la syntaxe est:

Syntaxe : **DISPOSE (**<identificateur>**) ;**

Où <identificateur> est le nom de la variable pointeur qui pointe l'espace à désallouer.

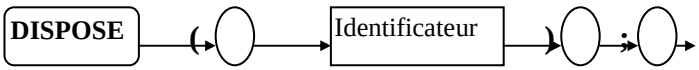


Diagramme de syntaxe

L'exécution de l'instruction DISPOSE (T), permet la désallocation de l'espace mémoire qui était occupé par la variable pointée T.

Exemple :

```
PROGRAM P_dispose;

TYPE  Art = RECORD
                Num: INTEGER;
                Car: CHAR;
              END;
        Point: ↑ Art;
VAR   P: Point;

BEGIN
NEW (P);     (* Allouer un espace mémoire *)

(* suite du programme : utilisation de P *)

DISPOSE (P);    (* Libérer l'espace *)

END.
```

4- Application sur les pointeurs :

La liste chaînée qui est une structure de données dynamique possède une taille variable, à tout moment des éléments peuvent être ajoutés (NEW) ou supprimés (DISPOSE).

Une liste est identifiée par le pointeur de son premier élément (qui est appelé sommet) et qui pointe sur le premier élément, ce dernier pointe sur l'élément suivant et ainsi de suite. Son dernier élément ne pointe nul part, il s'agit du pointeur NIL.

La structure d'un élément d'une liste est un article à plusieurs champs, dont l'un doit être obligatoirement de type pointeur et qui pointe sur l'élément suivant de l'élément en cours. C'est ce champ pointeur qui assure le chaînage entre deux éléments consécutifs d'une liste.

Soit à créer une liste chaînée dans laquelle nous sauvegardons toutes les lettres alphabétiques apparues dans un texte, ainsi que leur nombre d'apparitions (occurrences).

La structure d'une telle liste est déclarée par :

```
TYPE  lien = ↑ objet
        Objet = RECORD
                  Lettre : 'A'.. 'Z';
                  Occur: INTEGER;
                  Suivant: Lien;
                   END;
```

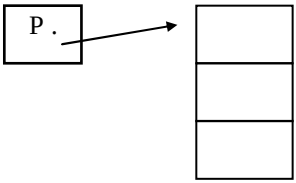
Supposons que nous voulions utiliser la variable pointeur nommée Sommet pour pointer le début de cette liste.

Initialement la liste est vide, aussi nous écrivons : Sommet : = NIL;

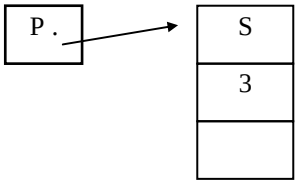
Nous avons besoin d'insérer maintenant un élément dans la liste. On crée un élément dynamiquement par la procédure NEW dont l'argument est un pointeur. Avec

```
VAR   P: Lien;
L'appel de NEW (P);
```

créera un élément de type Objet dont le nom est P↑. La figure suivante montre l'effet de cet appel.

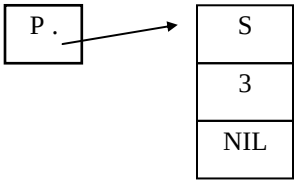


L'étape suivante est d'affecter les informations correspondantes au nouvel élément soit : P↑. Lettre := 'S';
(* si l'on suppose que la lettre 'S'est P↑. Occur:= 3; apparu 3 fois dans le texte traité *).
Le résultat de cette affectation est montré en figure :

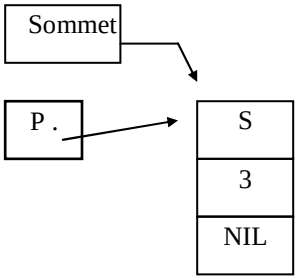


L'étape suivante, dont la raison deviendra rapidement évidente, est l'affectation P↑. Suivant := Sommet; (*)

La valeur courante de Sommet est NIL et l'effet de l'instruction (*) est de mettre P↑. Suivant à NIL, comme le montre la figure:



Enfin nous affectons Sommet := P pour obtenir le résultat désiré, montré par la figure :



Nous avons construit une liste contenant un élément : La valeur de Sommet↑. Lettre est 'S', de Sommet↑. Suivant est NIL.
Et de Sommet P. Occur est 3 et

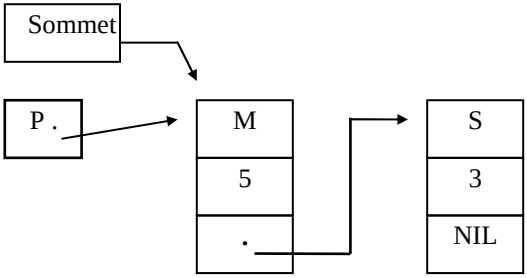
Nous pouvons insérer un autre élément dans la liste, de la même manière, on crée d'abord le nouvel élément :

```
NEW(P);
P↑. Lettre := 'M';
P↑. Occur := 5;
```

et faire ensuite, le chaînage avec le premier élément:

```
P↑. Suivant := Sommet;
Sommet := P;
```

Ce qui donne:



Maintenant sommet pointe l'élément nouvellement créé. Nous allons exprimer, l'opération de création de cette liste par un programme.

Supposons que Tabl est un tableau de caractères contenant les lettres alphabétiques de 'A' à 'Z' et Tabo un tableau d'entiers contenant les occurrences respectives des lettres alphabétiques apparues dans le texte traité.

```
PROGRAM Liste;
TYPE  Lien = ↑ Objet;
        Objet = RECORD
                  Lettre : 'A'.. 'Z';
                  Occur: INTEGER;
                  Suivant: Lien;
                  END;
VAR   Sommet, P: Lien;
        Tabl : ARRAY [1.. 26] OF CHAR;
        Tabo: ARRAY [1 .. 26 ] OF INTEGER;
        I: INTEGER;
BEGIN
Sommet: = NIL;
FOR I: = 1 TO 26
       DO
        BEGIN
                  NEW (P);
                  P ↑. Lettre:= Tabl [ I];
                  P ↑. Ocur:= Tabo [ I];
                  P ↑. Suivant:= Sommet;
                  Sommet:= P;
        END;
END.
```

IV- LES FICHIERS :

Les programmes que nous avons vus jusqu'ici ont tous provoqué une sortie et, le plus souvent ils ont aussi demandé une entrée. Ils les ont faits en utilisant les fichiers standards INPUT et OUTPUT. Dans ce paragraphe, nous allons parler en détails de ces fichiers et de bien d'autres.

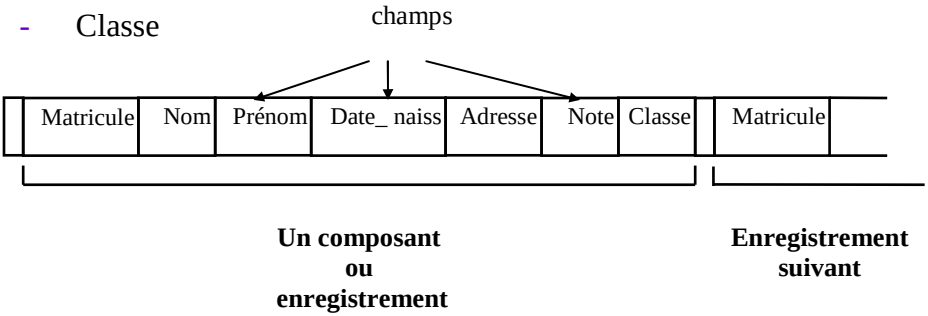
1- Définition :

Un fichier est un ensemble de données (champs) qui, regroupées, constituent une suite de composants de même type (appelés enregistrements) et stockés sur support externe (disque dur, disquette,...) autre que la mémoire centrale.

Exemple :

Un fichier des élèves dont les données sont :

- Matricule
- Nom
- Prénom
- Date de naissance
- Adresse
- Notes obtenues
- Classe



Où tout simplement il peut s'agir d'un fichier d'entiers, de réels ou d'un autre type prédéfini.

La déclaration d'un type fichier en Pascal, se fait au moyen des mots FILE OF dont la syntaxe est :

TYPE < identificateur > = FILE **OF** < Type elt >**;**

Où <Type-elt> est le type des composants du fichier, et qui peut être quelconque, sauf qu'il ne peut être lui-même fichier.

Diagramme de syntaxe

Exemple:

```
TYPE    Fich 1 = FILE OF INTEGER;
        Fich 2 = FILE OF BOOLEAN;
        Tab    = ARRAY [1..50] OF CHAR;
        Fich3 = FILE OF Tab;          ←——— fichier de tableaux
        Client = RECORD
                   Numcli : INTEGER;
                   Nom, Prénom : STRING [15];
                   adresse : STRING [40];
                   END;

VAR     Fentier : Fich1;
        Fbool   : Fiche 2;
        FTab    : Fiche 3;
        FPers   : FILE OF Client      ←\
        FVoiture : FILE OF Record     ←/ Fichiers d'articles
                     Numéro : STRING [8];
                     Marque  : STRING [10];
                     END;
```

- Fentier et Fbool sont des fichiers dont les composants sont des éléments simples.
- Ftab est un fichier de tableaux.
- Fpers et Fvoiture sont des fichiers d'articles.

Remarque :

1- Le fichier est un type de données homogène, tous ses composants sont de même type.
2- La longueur d'un fichier, c'est à dire le nombre d'enregistrements, n'est pas fixé au moment de la déclaration. Cependant, il dépend de la capacité de stockage du supoprt externe.
3- Contrairement à toutes les variables dont la durée de vie est limitée par celle du programme (il est chargé en mémoire centrale pour être exécuté, et dès que cette exécution est terminée, la mémoire est utilisée pour d'autres programmes), les variables de type fichier sont permanentes.
4- On peut stocker une plus grande masse de données dans un fichier que dans la mémoire centrale.
5- On ne déclare pas les fichiers standards INPUT et OUTPUT dans la partie déclaration des variables. INPUT doit apparaître dans l'entête d'un programme, si ce dernier utilise READ, READLN, EOF et EOLN sans nom de fichier en paramètres, OUTPUT doit apparaître dans l'entête d'un programme, si ce dernier utilise WRITE ou WRITELN, sans nom de fichier en paramètres.
6- Certains systèmes imposent de mentionner OUTPUT dans l'entête du programme même si le programme ne comporte aucun appel à WRITE, afin de pouvoir imprimer à coup sûr les messages d'erreurs.

2- Accès au fichier :

Comme un fichier est constitué d'un ensemble de composants identiques, l'accès à l'un d'entre eux se fait soit de manière séquentielle, c'est à dire qu'il y a eu déjà accès aux N composants qui le précédent, soit de manière directe, c'est à dire en indiquant le numéro d'ordre du composant concerné. Ce dernier cas est propre à certains compilateurs.

A chaque instant, il n'y a qu'un élément du fichier qui est accessible au programme. Le déplacement est activé par des instructions particulières, que nous allons voir ci-après.

L'accès à un des champs du fichier nécessite de désigner d'abord la variable fichier concernée, puis le champ concerné séparés par ↑., et dont la syntaxe ↓ est :

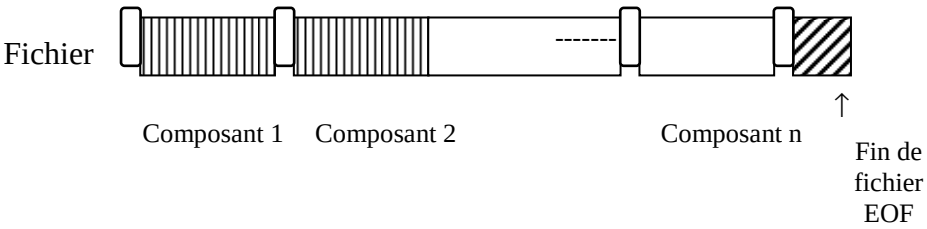
< Variable fichier > ↑ . < Idf champ >

< idf champ > peut lui même désigner un autre type structuré dans lequel est déclaré le champ désiré.

Exemple : PROGRAM Exemple (INPUT, OUTPUT);

```
TYPE      Adresse = RECORD
                      Num  : INTEGER;
                    Code : INTEGER;
                      END;
VAR   Cli : FILE OF RECORD
                        Nom, Prenom : STRING [15];
                      adr : Adresse;
                    END;
BEGIN
   .....
  READLN (Cli ↑. Nom);
   .....
 WRITELN (Cli ↑. Adr. Num);
    .....
END.
```

En mode séquentiel, les enregistrements sont créés ou lus les uns après les autres, l'ajout ne peut se faire qu'en fin de fichier, spécifiée par une identification particulier appelé EOF (End Of File c'est à dire Fin de fichier).



3- Création d'un fichier : REWRITE

L'appel de la procédure REWRITE permet de créer un nouvel fichier pour une éventuelle écriture ou d'ouvrir un fichier existant en écriture, en écrasant son contenu.

La syntaxe et la suivante :

REWRITE (< identification de fichier >**);**

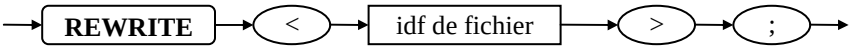


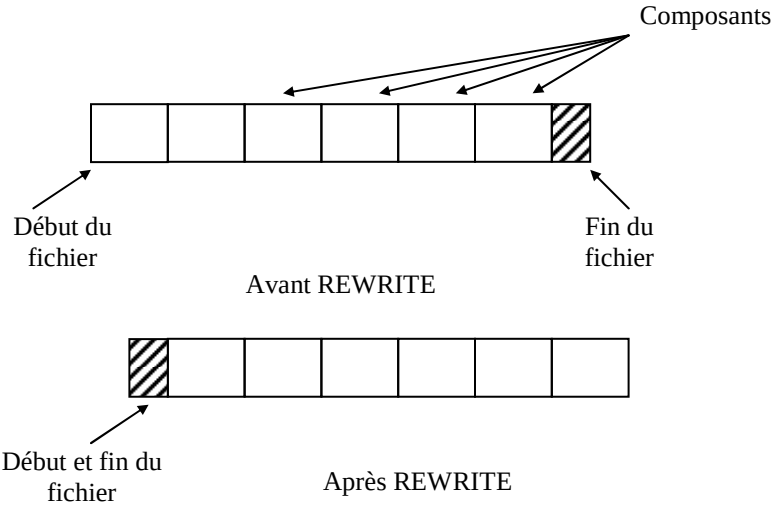
Diagramme de syntaxe

Nota : A identificateur de fichier qui est une variable de type fichier déclaré dans le programme peut être ajouté le nom du fichier sur support extérieure suivi par l'extension. DATA pour spécifier qu'il s'agit d'un fichier de données.

Exemple :

REWRITE ($Fich_1$, '$Fich_1$. DATA');

Si le fichier existait préalablement, l'ensemble de ses composants seraient perdus.



Après l'ordre REWRITE, le début et la fin de fichier sont au même emplacement.

Remarque : L'indicateur EOF précise la fin de fichier et sa valeur est de type booléen. Il est créé automatiquement après la création du dernier composant. Son utilisation permet de détecter l'endroit où il est possible d'ajouter de nouveaux composants ou s'il reste des composants à lire.

Syntaxe : **EOF (**< idf de fichier >**)**

Exemple :

WHILE NOT (EOF ($Fich_1$))

DO .....

Ce qui signifie que tant que la fin de fichier $Fich_1$ n'est pas détectée alors faire ...

4- Écriture dans un fichier : PUT / WRITE

Après l'instruction REWRITE, qui a eu pour objet de se positionner au début du fichier, il faut écrire les données, qui se trouvent dans le tampon de la mémoire du fichier concerné, sur le support externe à l'aide de :

1- l'instruction PUT.

PUT (< idf fichier >**) ;** **Syntaxe** :

Ceci a pour effet de créer un nouvel élément dans le fichier à la position indiquée par le marqueur et de déplacer celui-ci d'un cran à droite.

2- l'instruction WRITE.

WRITE (< idf fichier >**;** < tampon mémoire >**) ; Syntaxe :**

Supposons que l'on ait déclaré :

VAR Tampon : Client **(**Client est le type article déclaré précédemment**)**.

Et que Tampon, contienne des informations à écrire dans le fichier Fpers. Alors les ordres :

```
Fperst ↑ := Tampon;
PUT (Fpers);
```

Peuvent être abrégés en :

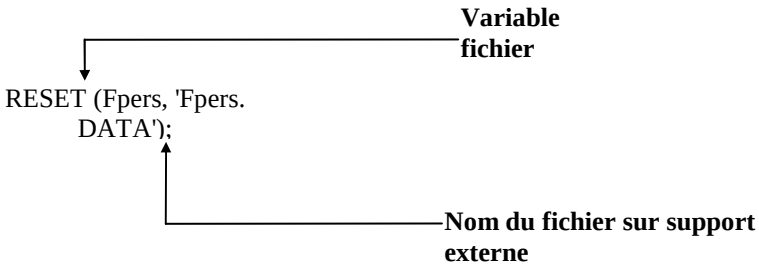
```
WRITE (Fpers, Tampon);
```

5- Ouverture d'un fichier en lecture : RESET

Avant toute manipulation de fichiers, il est obligatoire de procéder à leur ouverture, c'est à dire de positionner la fenêtre sur le début du fichier qui peut être aussi sa fin dans le cas d'une création. L'instruction concernée est RESET et permet d'effectuer une ouverture de fichier en lecture.

Syntaxe : **RESET (<** idf fichier **>);**

Exemple :



6- Lecture d'un fichier : GET, READ

L'instruction GET a pour objet de positionner la fenêtre sur le composant suivant et de le mettre dans zone tampon.

Syntaxe : **GET (**idf fichier**);**

Exemple : GET (Fpers);

Remarque :

1- Après l'appel de GET, on est dans l'un des deux cas suivants :

EOF (Fpers) = FALSE et Fpers↑ contient l'élément suivant

Ou

EOF (Fpers) = TRUE et Fpers↑ est indéfini (il s'agit d'une erreur système).

2- S'il arrive que le fichier soit vide au moment de lire le premier enregistrement, alors EOF (Fpers) sera égal à True immédiatement après l'appel de RESET. C'est pour cette raison qu'il vaut mieux tester EOF avant d'appeler GET.

La procédure READ, dont la syntaxe est READ (idf fichier), (idf tampon); sert aussi à lire un fichier, si l'on écrit :

READ (Fpers, Tampon) elle est équivalente à :

tampon : = Fpers ↑;
GET (Fpers);

Si EOF rend la valeur true après la lecture, tampon contient alors le dernier élément du fichier.

APPLICATION :

Les fichiers, à la différence de variables d'autres types, ne peuvent être recopiés qu'élément par élément. Le programme Cpaquet suivant a pour objet de copier un paquet de cartes en lisant sur le fichier Cartein et en écrivant sur le fichier Carteout. Les cartes vierges de carte in ne sont pas recopiées sur carte out.

```
PROGRAM Cpaquet (Cartein, Carteout, OUTPUT);
CONST  Blanc= ' ';

TYPE    Indice = 1..' 80;
        Carte = PACKED ARRAY [indice] OF CHAR;
        Fcarte = FILE OF Carte;

VAR     Cartein, Carteout : Fcarte;
        Tampon, Cvierge : Carte;
        l : Indice;

BEGIN
FOR     l : = 1   TO 80
    DO  Cvierge [l]: = Blanc;
RESET (Cartein);

REWRITE (Carte out);
WHILE NOT ( EOF(Cartein) )
    DO  BEGIN  READ (Cartein, Tampon);
      IF Tampon <> C vierge
THEN WRITE (Carte out, Tampon).
END;
END.
```

7- Les fichiers texte :

Le type standard TEXT, est défini par une suite de caractères organisés en lignes. Chaque ligne est de longueur variable mais est terminée par un caractère spécial de fin de ligne (EOLN pour End Of Line).

Il existe deux fichiers texte standards en Pascal : INPUT et OUTPUT et qui correspondent à :

```
TYPE    Line = PACKED ANAY [1..80] OF CHAR;
                Text = FILE OF Line;

VAR     INPUT, OUTPUT : Text;  ◄—— Type prédéfini
```

Le mot utilisé pour spécifier un fichier texte est TEXT.

Exemple:

```
VAR   Fich1 : FILE OF CHAR;
      Fich2 : TEXT;
```

Ici, $Fich_1$ est un fichier de composants de type CHAR;
$Fich_2$ est un fichier de texte, structure en lignes.

Nous avons déjà utilisé, dès les premiers chapitres, deux opérations propres aux fichiers de texte, il s'agit de : READ et WRITE.

WRITE (Fich, Elm) : écrit l'élément dans le fichier Fich.
READ (Fich, Elm) : lit l'élément et à partir du fichier Fich.

Il existe d'autres opérations dont certains sont spécifiques à certains compilateurs. Cependant, il existe deux standards :

* EOLN: Teste la fin de ligne, retourne true ou false.
*PAGE: Provoque un saut de page sur écran ou sur imprimante.

Exemple :

PAGE (OUT PUT); provoque le saut de page d'écran

Remarque :

READ (Car); est équivalent à READ (INPUT, Car);
WRITE (Car); est équivalent à WRITE (OUTPUT, Car);

V- RÉSUMÉ :

Dans cette leçon nous avons vu les différents types structurés de données que le langage Pascal permet d'utiliser.

- Le type **ensemble** défini par SET OF .... est un nombre fini de valeurs de même type. Ces derniers peuvent être de type caractère, entier, booléen, intervalle ou type énuméré.

- Le type **ARTICLE** défini par RECORD .... END; est constitué de plusieurs champs qui peuvent être de types différents et qui sont accessibles directement par leur nom. Il existe deux types d'articles en Pascal: Les articles sans variantes et les articles avec variantes, dans lesquels un ensemble de champs varient d'un enregistrement à l 'autre.
- Le type **pointeur** n'est pas un identificateur prédéfini, mais le contenu d'une variable de type pointeur est une adresse assurant l'accès à un élément dynamique.

- Le type **fichier** FILE OF ... permet de définir une collection d'objets de même type et qui peuvent être conservés en permanence sur des supports externes. Un élément du fichier est appelé enregistrement, il peut être de type quelconque sauf fichier. Il existe deux types de fichiers : Les fichiers de données et les fichiers textes qui sont organisés en un nombre de lignes, constitués à leur tour d'un ensemble de caractères.

EXERCICES CORRIGÉS :

EXERCICE N°01 :

Ecrire un programme qui permet de tester si les caractères entrés au clavier font partie de l'ensemble des caractères alphabétiques. La fin de l'entrée est détectée par la frappe du caractère ! (point d'exclamation).

EXERCICE N°02 :

Donner la déclaration de Pascal l'enregistrement dans la structure est donnée ci-après : **article membre**

NOM DU CHAMP	OBJET	TYPE DE DONNEE
Num	numéro	entier
Nom	nom de personne	20 caractères
PRE	prénom	10 caractères
ADR	adresse	
Num	numéro de rue	3 caractères
Rue	nom de rue	30 caractères
COD	code postal	5 caractères
Vil	nom de ville	15 caractères
CLU	code utilisateur	3 caractères
DAT	date d'entrée	JJ/MM/AA 8 caractères
DER	dernier paiement acquitté	booléen
LOT	montant de la cotisation	entier

Écrire le programme qui permet de créer un tel article. Vérifier l'information par sa lecture

EXERCICE N°03 :

Un élément géométrique simple peut être un point, une ligne ou un cercle.

- Un point est défini par la donnée de ses cordonnées (sur l'axe des X et sur l'axe des Y) ;
- Une ligne est définie par la donnée du coefficient des X, du coefficient des Y et d'une constante ;
- Un cercle est défini par la donnée des coordonnées de son centre et la taille de son rayon.

Exemple : point : (12, -4)
ligne : 4 x + 3 y - 5 = 0
cercle : centre (0,0) rayon 5.

Donner la déclaration Pascal de la structure de l'article élément géométrique.

EXERCICE N°04 :

Écrire un programme qui lit une suite de caractères, les stocke dans une liste chaînée et les imprime en sens inverse. La fin de la chaîne est identifiée par le caractère ' #' .

EXERCICE N°05 :

Écrire un programme qui lit la liste des candidats d'un examen donné ainsi que les moyennes obtenues et les classe dans trois fichiers différents : le premier contient les candidats qui ont réussi l'examen avec une moyenne supérieure à 10,5 , le second contient ceux qui l'ont réussi avec une moyenne comprise entre 9,80 et 10,5 (liste d'attente) le troisième contient ceux qui l'ont échoué (moyenne inférieure ou égale à 9,80). La liste se termine par la donnée d'un nom vide.

EXERCICE N°06 :

Écrire un programme qui effectue le comptage de la fréquence de lettres dans une ligne entrée au clavier et écrit le résultat dans un fichier texte de la forme :

1ère ligne : Voici la fréquence d'apparitions des lettres

2ème ligne :	Lettre	Fréquence
3ème ligne :	A	23
4ème ligne :	B	2
.		
.		
28ème ligne :	Z	4

CORRIGÉS DES EXERCICES

EXERCICE N°01 :

```
PROGRAM Alpha (INPUT, OUTPUT);
VAR   Letters: SET OF 'A'.. 'Z';
      Tab: ARRAY ['A'.. 'Z'] OF INTEGER;
      C: CHAR;
      I, J: INTEGER;
```

Initialisation de l'ensemble Lettres

```
BEGIN
Letters: = ['A'.. 'Z']
I: = 0; J: =0;
FOR   C: = 'A' TO 'Z'
      DO     Tab [C]:= 0;
C :='*' ;
WRITELN ('Entrer les caractères') ;

WHILE (C <> ' !')
BEGIN              DO
READ (C);
IF C IN Letters
THEN BEGIN
Tab [C]:= Tab [C] + 1;
I: = I+1;
END
ELSE   J=J+1;
END;
FOR C: = 'A'   TO 'Z'
WRITELN (C, ':', Tab[C]);      DO

WRITELN;
WRITELN (I, 'letters'); WRITELN;
WRITELN (J, 'autres caractères') ;
END.
```

Comptage des caractères non inclus dans Lettres

EXERCICE N°02:

```
PROGRAM Declare;
TYPE  Adres = RECORD
Num : STRING [3] ;
Rue: STRING [30];
Cod: STRING [5];
Vil: STRING [15] ;
END;
VAR   Membre : RECORD
Num : INTEGER ;
Nom: STRING [20];
Pré : STRING [10] ;
Adr : Adres ;
Clu : STRING [3] ;
Dat : STRING [8] ;
Der: BOOLEAN;
Cot: INTEGER;
END;

PROGRAM  Créer (INPUT, OUTPUT) ;
(* partie déclaration de Membre *) ;
Car : CHAR ;

BEGIN
WRITE (‘ Numéro de membre’) ; READLN (Membre. Num) ;
WRITE (‘ Nom de membre’) ; READLN (Membre. Nom) ;
WRITE (‘ Prénom de membre’) ; READLN (Membre. Pré) ;
WRITE (‘ l’adresse  (Numéro de la rue)’) ; READLN (Membre. Adr.
Num) ;
WRITE (‘ l’adresse  (Nom de la rue)’) ; READLN (Membre. Adr.
Rue ) ;
WRITE (‘ le code postal’) ; READLN ( Membre. Adr. Cod ) ;
WRITE (‘ l’adresse  (Nom de la ville)’ ) ; READLN ( Membre.
Adr.Vil ) ;
WRITE (‘ Code utilisateur’) ; READLN ( Membre. Clu ) ;
WRITE (‘ Date d’entrée ( JJ/MM/AA)’) ; READLN (Membre. Dat) ;
WRITE (‘Dernier mois payé (O/N)’) ; READLN (Car) ;
```

```
IF Car = 'O'
THEN Member. Der := TRUE
ELSE  Membre. Der := FALSE ;
WRITE (' Montant de cotisation') ; READLN (Membre. Cot) ;
WRITELN; WRITELN;
WRITE ('Numéro', Membre. Num ) ;
WRITE (' Nom', Membre. Nom) ;
WRITE (' Prénom', Membre. Pré) ;

WRITE (' Numéro de la rue', Membre. Adr. Num) ;
WRITE (' Nom de la rue', Membre. Adr. Rue) ;
WRITE (' le code postal', Membre. Adr. Cod) ;
WRITE (' Numéro de la rue ', Membre. Adr.Num ) ;
WRITE (' Code utilisateur',  Membre. Vil) ;
WRITE (' Date d'entrée', Membre. Dat) ;
WRITE ('Dernier mois payé, ' Membre. Der) ;
WRITE ( ' Montant de cotisation', Membre . Cot ) ;
END.
```

EXERCICE N°03 :

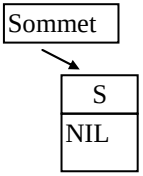
```
Géométrique ; PROGRAM
Coordo = RECORD TYPE
X, Y: REAL;
END ;
Forme = (Point, Ligne, Cercle) ;
Elément = RECORD
CASE   E_geom : Forme  OF
Point: ( Pos : Coordo ) ;
Ligne: ( Xcoeff, Ycoeff, Cons : REAL ) ;
Cercle: (Centre : Coordo ; Rayon : REAL) ;
END ;
```

EXERCICE N°04:

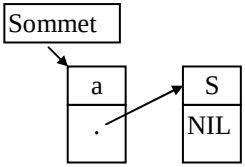
```
PROGRAM Listeinv (INPUT, OUTPUT ) ;
TYPE  lien = ↑ Objet ;
        Objet = RECORD
                  Info : CHAR ;
                  Suivant : Lien ;
                   END ;

VAR   Sommet, P, Q : Lien ;
        Car: CHAR;
BEGIN
Sommet: = NIL;

WHILE   Car <> '#'
          DO BEGIN
                  NEW (P);
                  READ (P↑ . Info) ;
                  P ↑ . Suivant := Sommet ;
                  Sommet := P ;
                 END;
P: = Sommet;

WHILE P <> NIL
        DO    BEGIN
                  WRITE (P ↑ . Info); Q = P;
                  P := P ↑ . Suivant ; DISPOSE (Q)
                 END ;

END.
```

Si la chaîne lue est par exemple Salut#

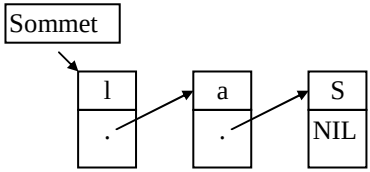
La création de la liste se fait par les étapes de (1) à (5) suivantes :



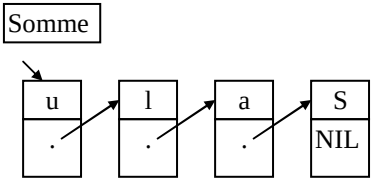
(1)



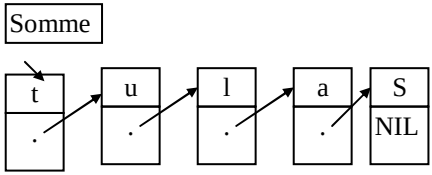
(2)



(3)

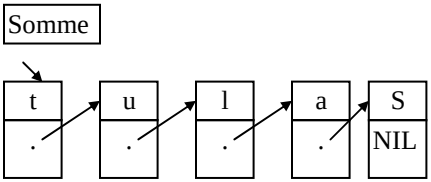


(4)



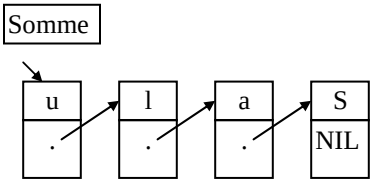
(5)

L'impression de la liste se fait par les étapes de (1) à (5) suivantes :



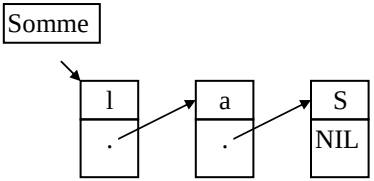
Impression de t

(1)



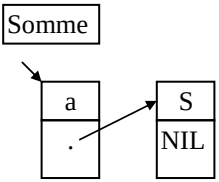
Impression de u

(2)



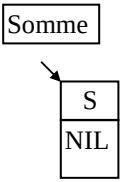
Impression de l

(3)



Impression de a

(4)



Impression de S

(5)

À chaque pas on libère un élément de la liste, à la fin de la liste directement.

EXERCICE N°05:

```
PROGRAM   Candidat (INPUT);
TYPE  Cand = RECORD
Nom, Prénom: STRING [15];
Moy: REAL;
END;
VAR   Bon, Moyen, Faible: FILE OF Cand;
Tampon: Cand;
REWRITE (Bon, 'Bon .DATA');
REWRITE (Moyen, 'Moyen .DATA');
REWRITE (Faible, 'Faible. DATA');

BEGIN
Tampon. Nom: = 'a';
WHILE Tampon. Nom <> '' ''
DO   BEGIN
WRITE ('nom du candidat'); READLN (Tampon. Nom);
WRITE ('prénom du candidat'); READLN (Tampon. Prénom);
WRITE ('moyenne obtenue'); READLN (Tampon.  Moy);
IF   Tampon .Moy > 10.5
THEN WRITE (Bon, Tampon)
ELSE  IF 9.80 < Tampon .Moy <= 10.5
THEN WRITE (Moyen, Tampon)
ELSE  WRITE (Faible, Tampon);
END;
END.
```

EXERCICE N°06:

```
PROGRAM Freq (INPUT, OUTPUT);
VAR   Car: CHAR;
Tab: ARRAY [‘A’ .. ‘Z’] OF INTEGER;
Lettres: SET OF ‘A’.. ‘Z’;
Sortie: TEXT;
BEGIN
FOR   Car: = ‘A’ TO ‘Z’
DO   Tab [Car]:= 0;
WRITELN (‘Entrée une ligne’);
WHILE NOT (EOLN)
DO     BEGIN
READ (Car);
IF Car IN Lettres
THEN Tab [Car]:= Tab [Car] + 1;
END;
REWRITE (Sortie, ‘Sortie. TXT’);
WRITELN (Sortie, ‘Voici la fréquence d’apparition des lettres’);
WRITELN (Sortie, ‘ Lettre ---------- Fréquence’);
FOR   Car := ‘A’   TO ‘Z’
DO     WRITELN (Sortie, Car, ‘----------’, Tab [Car]);
CLOSE (Sortie);
END.
```